Конспект лекции
Инструменты сборки

Цель и задачи лекции

Цель – изучить инструменты сборки проектов и их функционал.

Задачи:

1. Дать понимание термину сборка проекта
2. Изучить устройство Maven и его функционал
3. Ознакомиться с преимуществами Gradle

План занятия

1. Понятие сборки
2. Maven
3. Gradle

Понятие сборки

Сборка (англ. assembly) — двоичный файл, содержащий исполняемый код программы или (реже) другой подготовленный для использования информационный продукт.

Автоматизация сборки — этап написания скриптов или автоматизация широкого спектра задач применительно к ПО, применяемому разработчиками в их повседневной деятельности, включая такие действия, как:

1. компиляция исходного кода в бинарный код
2. сборка бинарного кода
3. выполнение тестов
4. разворачивание программы на производственной платформе
5. написание сопроводительной документации или описание изменений новой версии

Целью сборки называется конечный файл (файлы) получаемые в процессе сборки.

Простые проекты можно собрать в командной строке. Если собирать большие проекты из командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от зависимости от платформы и упростить написание скрипта используют инструменты для сборки проекта.

Maven

Maven – это инструмент для сборки проектов. Maven позволяет:

- собирать проект
- управлять документацией
- создавать отчеты
- подключать зависимости
- управлять релизами
- работать с системами контроля версий (SCM)
- управлять поставкой сборки

Основные преимущества Maven:

- Независимость от OS. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.
- Управление зависимостями. Редко какие проекты пишутся без использования сторонних библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Maven позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.
- Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (Continuous Integration).
- Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты, которые собираются с помощью maven. При этом зачастую проект настраивать не нужно - он сразу готов к дальнейшей разработке. Как следствие - если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.
- Декларативное описание проекта. Maven может обеспечить выгоды для процесса сборки, используя стандартные правила и практику, чтобы ускорить цикл разработки, в то же время, помогая вам достичь более высокий уровень успеха.

Maven решает эти проблемы путем стандартизации структуры папок и внутреннего устройства проекта. Maven дает рекомендации, где должны находиться различные части проекта, такие, как исходный код, код тестов и конфигурационные файлы. Например, Maven предполагает, что весь исходный код Java должен быть размещен в папке src\main\java. Всё это облегчает понимание и навигацию по любому проекту Maven.

Декларативное управление зависимостями

Большинство проектов Java для правильного функционирования полагаются на другие проекты и фреймворки с открытым кодом. Ручное скачивание этих зависимостей и поддержка их версий при работе над проектом может оказаться довольно затруднительным делом.

Maven предоставляет удобный способ объявить зависимости проекта в отдельном, внешнем файле с именем pom.xml. После чего Maven

автоматически загружает эти зависимости и позволяет вам использовать их в своем проекте. Это значительно упрощает управление зависимостями проекта. Важно отметить, что в файле pom.xml вы указываете, что зависит, а не как. Кроме того, файл pom.xml выступает в качестве инструмента документирования, сообщая о зависимостях вашего проекта и его версии.

Плагины

Maven использует архитектуру, основанную на плагинах, что делает его легко расширяемым, а его функциональность – легко настраиваемой. Эти плагины инкапсулируют многократно используемые алгоритмы сборки и выполнения задач. Сегодня существуют сотни доступных Maven-плагинов, которые могут быть использованы для выполнения широкого круга задач, от компиляции кода до генерации проектной документации.

Кроме того, Maven позволяет легко создавать свои собственные плагины, тем самым позволяя вам выполнять задачи и процессы, специфичные для вашей организации.

Единая абстракция сборки

Maven обеспечивает единый интерфейс для сборки проектов. Вы можете построить Maven-проект используя всего несколько команд. После того, как вы познакомитесь с процессом сборки Maven, разработчик без труда поймет, как строить и другие проекты Maven. Это освобождает разработчиков от необходимости изучать индивидуальные особенности построения, позволяя таким образом сосредоточиться на разработке.

Поддержка инструментов

Maven предоставляет мощный инструмент командной строки выполнения различных операций. Сегодня все основные IDE предоставляют прекрасную поддержку инструментов для Maven. В добавок, Maven полностью интегрирован с такими современными инструментами непрерывной разработки, как Jenkins, Bamboo, и Hudson.

Архетипы

Как мы уже упоминали, Maven обеспечивает стандартное расположение папок для своих проектов. Когда приходит время создавать новый проект Maven, вам приходится вручную создавать каждую папку, что может быстро стать утомительным. И это тот самый момент, когда архетипы Maven приходят на помощь. Архетипы Maven являются предопределенными шаблонами проектов, которые могут быть использованы для создания новых проектов. Проекты, созданные с помощью архетипов, будут содержать все папки и файлы, необходимые для работы.

Кроме того, архетипы являются ценным инструментом для хранения лучших методик и совместных активов, которые будут нужны в каждом проекте. Архетипы также являются ценным инструментом для обобщения

лучших практик и общих активов, которые вам понадобятся в каждом из ваших проектов.

Создание проекта Maven

Чтобы создать простой проект Maven, необходимо выполнить команду:

```
mvn -B archetype:generate \

      -DarchetypeGroupId=org.apache.maven.archetypes \

      -DgroupId=com.mycompany.app \

      -DartifactId=my-app
```

Выполнив эту команду, вы заметите, что произошло несколько вещей. Во-первых, для нового проекта был создан каталог с именем my-app , и этот каталог содержит файл с именем pom.xml, который должен выглядеть следующим образом:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>Maven Quick Start Archetype</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

pom.xml содержит объектную модель проекта (POM) для этого проекта. POM является основной единицей работы в Maven. Это важно помнить, потому что Maven изначально ориентирован на проект, поскольку все вращается вокруг понятия проекта. Короче говоря, POM содержит всю важную информацию о вашем проекте и, по сути, является универсальным средством поиска всего, что связано с вашим проектом.

Основные элементы Maven

Основные элементы файла pom.xml:

- project – этот элемент верхнего уровня во всех файлов pom.xml Maven.
- modelVersion – этот элемент указывает, какую версию объектной модели этого POM использует.

- groupId – этот элемент указывает на уникальный идентификатор организации или группы, которые создали проект. GroupId является одним из ключевых идентификаторов проекта и, как правило, основаны на полное доменное имя вашей организации. Например org.apache.maven.plugins является назначенным GroupId для всех Maven плагинов.
- artifactId – этот элемент указывает на уникальное имя базы первичных артефактов, генерируемых этим проектом. Основной артефакт для проекта, как правило, JAR-файл. Типичный артефакт производства Maven будет иметь вид <artifactId> -. <version> <extension> (Например,MyApp-1.0.jar).
- packaging – этот элемент указывает на тип пакета, который будет использоваться этот артефакт (например, JAR, WAR, EAR и др.). Это не только означает, что если артефакт произведенного JAR, WAR, или EAR, но также может указывать на конкретного жизненного цикла для использования в качестве части процесса сборки. EAR – это если нам от какого-либо проекта не нужен jar файл (к рутовый проект который содержит в себе лишь описание). В зависимости от того какой тип вы выберете, такая сборка и будет (Сборка для веб приложение отличается от настольного ...).
- version – этот элемент указывает на версию артефакта, генерируемых проектом. Maven проходит долгий путь, чтобы помочь вам в управление версиями, и вы часто будете видеть SNAPSHOT обозначение в версии, которая указывает, что проект находится в состоянии развития.
- name – этот элемент указывает отображаемое имя для проекта. Это часто используется в генерации документации Maven (Это также может быть имя проекта в Nenbeans).
- url – этот элемент указывает, где можно найти сайте проекта. Это часто используется в генерации документации Maven.
- description – этот элемент представляет собой общее описание вашего проекта. Это часто используется в генерации документации Maven.

После генерации архетипа проекта будет создана следующая структура каталогов:

```
my-app

|-- pom.xml

`-- src

    |-- main

    |   `-- java

    |       `-- com

    |           `-- mycompany
```

```
|               `-- app
|                   `-- App.java
`-- test
    `-- java
        `-- com
            `-- mycompany
                `-- app
                    `-- AppTest.java
```

Как видно из вывода, скомпилированные классы были помещены в ${basedir}/target/classes, что является еще одним стандартным соглашением, используемым Maven.

Обозначений версий Maven

В проекте Maven рекомендуется использовать следующее соглашение для обозначения версий:

```
<major-version>.<minor-version>.<incremental-version>-qualifier
```

Старшее (major), младшее (minor) и дополняющее (incremental) значения являются числами, а квалификатор (qualifier) принимает такие значения, как RC, alpha, beta и SNAPSHOT. Примеры нумерации, следующей этому соглашению: 1.0.0, 2.4.5-SNAPSHOT, 3.1.1-RC1 и т.д.

Квалификатор SNAPSHOT в версии проекта имеет специальное значение. Он показывает, что проект находится в стадии разработки. Когда проект использует зависимость SNAPSHOT то при каждом построении проекта Maven будет получать и использовать самый последний из артефактов SNAPSHOT.

Большинство менеджеров хранилищ принимают сборку релиза лишь единожды. Однако, если вы разрабатываете приложение в среде с непрерывной интеграцией, то производить сборки вам нужно часто и в менеджере хранилища размещать последнюю из них. Следовательно, суффикс SNAPSHOT является наилучшим способом пометить версию, находящуюся в процессе разработки.

Цели и плагины

Процессы построения, генерирующие артефакты, обычно требуют нескольких шагов и заданий для успешного завершения. Примеры таких задач включают компилирование исходного кода, запуск модульного теста и упаковка артефактов. Maven использует концепцию целей для представления таких гранулированных задач.

Для лучшего понимания того, что представляют собой цели, рассмотрим пример. Листинг отображает цель compile, выполненную над кодом проекта gswm в каталоге C:\apress\gswmbook\chapter5\gswm. Как и подразумевает само название, цель compile компилирует исходный код. Цель compile обнаруживает в папке src\main\java Java-класс HelloWorld.java, компилирует его и размещает скомпилированный класс в папке target\classes.

```
C:\apress\gswm-book\chapter5\gswm>mvn compiler:compile

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building Getting Started with Maven 1.0.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-compiler-plugin:2.5.1:compile (default-cli) @ gswm --

[WARNING] File encoding has not been set, using platform encoding Cp1252,

i.e. build is platform dependent!

[INFO] Compiling 1 source file to C:\apress\gswm-book\chapter5\gswm\target\ classes

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 1.197s

[INFO] Finished at: Mon Oct 13 22:11:42 MDT 2014 [INFO] Final Memory: 7M/18M

[INFO] ------------------------------------------------------------------------
```

Цели в Maven упакованы в плагины, которые по сути являются коллекциями из одной или более целей. В Листинге maven-compiler-plugin - это плагин который предоставляет цель compile. Как уже упоминалось ранее, директория target содержит сгенерированные Maven временные файлы и артефакты. Случается, что директория target становится слишком большой или из неё нужно удалить некоторые кэшированные файлы. Цель clean выполняет именно это, пытаясь удалить папку target вместе со всем её содержимым:

```
mvn clean:clean
```

Слово clean перед двоеточием обозначает плагин clean, а clean после двоеточия обозначает цель clean. Теперь становиться очевидным, почему запуск цели из командной строки требует такого синтаксиса:

```
mvn plugin_identifier:goal_identifier
```

Плагины и их поведение могут быть сконфигурированы с использованием секции <plug-in> файла pom.xml. Рассмотрим случай, когда вам нужно заставить проект компилироваться с помощью Java 1.6. Но в версии 3.0 плагин maven-compiler-plugin компилирует код с использованием Java 1.5. Поэтому необходимо модифицировать поведение этого плагина в файле pom.xml.

```
...
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
...
```

Теперь если вы выполните команду mvn compiler:compile, то сгенерированные файлы класса будут версии Java 1.6.

Хранение ресурсов в jar-файле

Из-за декларативного описания Maven, можно упаковать ресурсы в JAR-файлы просто путем размещения этих ресурсов в стандартную структуру каталогов. Любые каталоги или файлы, помещенные в ${basedir}/src/main/resources каталога упакованы в JAR с точно такой же структуры, начиная с основания JAR.

Gradle

Gradle (http://gradle.org/) является новейшим дополнением к семейству инструментов автоматизации процессов сборки Java-проектов. В отличие от Ant или Maven, использующих XML для конфигурирования, Gradle использует основанный на Groovy предметно-ориентированный язык Domain Specific Language (DSL).

Gradle обеспечивает гибкость, как у Ant, и использует точно такое же представление задач. Кроме того, он следует соглашениям и стилю управления зависимостями, как у Maven. В Листинге показано содержимое файла build.gradle по умолчанию.

```
apply plugin: 'java'

version = '1.0'

repositories {

mavenCentral()

}

dependencies{

testCompile group: 'junit', name: 'junit', version: '4.10'

}
```

Предметно-ориентированный язык DSL системы сборки Gradle и то, что он следует принципам CoC, являются причиной компактности gradle-файлов сборки. Первая строка в Листинге содержит указание на плагин Java, используемый для построения. Плагины в Gradle обеспечивают проект предварительно сконфигурированными задачами и зависимостями. Плагин Java, например, обеспечивает задачи для сборки файлов исходного кода, запуска юнит-тестов и установки ресурсов. Секция dependencies файла указывают Gradle использовать зависимости JUnit при компиляции исходных файлов тестов. Гибкость Gradle, как и гибкость Ant, в случае злоупотребления может стать причиной возникновения тяжелых и сложных сборок.

Литература и ссылки

1. http://maven.apache.org/guides/getting-started/index.html
2. Balaji Varanasi, Sudha Belida. Introducing maven. Apress, 2014

Вопросы для самоконтроля

1. Что такое сборка?
2. В чем ключевое отличие Maven и Gradle?